Parallel Exploration of the Nuclear Chromosome Conformation with NuChart-II

Parallel Exploration of the Nuclear Chromosome Conformation with *NuChart-II*

Fabio Tordini*, Maurizio Drocco*, Claudia Misale*, Luciano Milanesi‡, Pietro Lió†, Ivan Merelli‡, Marco Aldinucci*
*Computer Science Department, University of Turin, Italy.
tordini@di.unito.it, drocco@di.unito.it, aldinuc@di.unito.it
†Computer Laboratory, University of Cambridge, UK.
pietro.lio@cl.cam.ac.uk
‡Institute for Biomedical Technologies - Italian National Research Council, Segrate (Mi), Italy.
luciano.milanesi@itb.cnr.it, ivan.merelli@itb.cnr.it

***Abstract*—High-throughput molecular biology techniques are widely used to identify physical interactions between genetic elements located throughout the human genome. Chromosome Conformation Capture (3C) and other related techniques allow to investigate the spatial organisation of chromosomes in the cell's natural state. Recent results have shown that there is a large correlation between co-localization and co-regulation of genes, but these important information are hampered by the lack of biologists-friendly analysis and visualisation software. In this work we introduce NuChart-II, a tool for Hi-C data analysis that provides a gene-centric view of the chromosomal neighbourhood in a graph-based manner. NuChart-II is an efficient and highly optimized C++ re-implementation of a previous prototype package developed in R. Representing Hi-C data using a graph-based approach overcomes the common view relying on genomic coordinates and permits the use of graph analysis techniques to explore the spatial conformation of a gene neighbourhood.**

Keywords*—*Systems Biology, Parallel Computing, Hi-C data, Neighbourhood Graph, Chromosome Conformation Capture

I. Introduction

The three-dimensional organization of chromosomes and the physical interactions occurring along and between them play an important role in the regulation of gene activity. The spatial organization of chromatin is involved in compartmentalizing the nucleus and bringing widely separated functional elements into close spatial proximity. This structural conformation of the DNA in the nucleus is of critical importance for the gene regulation, in particular while analysed in correlation with other epigenetics effects.

Over the last decade, a series of molecular and genomic approaches have been developed to study three-dimensional chromosome folding at increasing resolution and throughput [1]; these methods are all based on Chromosome Conformation Capture (3C) and allow the determination of the frequency with which any pair of *loci* in the genome is in close enough physical proximity (probably in the range of 10-100 nm) to become cross-linked [2]. The three-dimensional (3D) conformation of chromosomes is involved in compartmentalizing the nucleus and bringing widely separated functional elements into close spatial proximity. Among 3C-based techniques, *Hi-C* is a method that adapts the above approach in order to exploit Next-generation sequencing, providing read pairs correspondence to genomic loci that physically interact in the nucleus [3]. The output of a Hi-C process is a list of pairs of locations along the chromosome, which can be represented as a square matrix Y, where $Y_{i,j}$ stands for the sum of read pairs matching in position i and position j, respectively. This matrix-based representation, called *contact map*, is reliable while looking at the interactions between two chromosomes, but becomes unsuitable to describe the neighbourhood of a gene or a cluster of genes. On the other hand, the possibility of creating a graph-based representation of Hi-C data can be very useful to create a map on which other omics-data can be mapped in order to characterize the different spatially associated domains.

Representing a genome as a graph gives the possibility to change the point of view, focusing on the cooperation among genes which can in turn be interpreted using statistical tools common in graph theory and network analysis. Common measurements, like *centrality* index analysis, permit to describe the neighbourhood of each node of the graph (i.e. genes), so as to evince information on the most influential gene of the network. Community analysis is another interesting study over the resulting graph since it can provide valuable information concerning genes interaction. Community analysis includes the detection of cliques and clusters in the network of genes.

Although several tools for Hi-C data processing exist, most of them propose contact maps for the analysis of chromatin structure (see section II-A). *NuChart* [4] proposed a graph-based approach: it was initially implemented as an R package designed to describe the chromosomal neighbourhood by integrating Hi-C data and information about genes position. NuChart works directly with sequenced reads to identify the related Hi-C fragments and, consequently, the genes containing those fragments, considering both intra-genic and inter-genic cases. The implementation, relying on the R environment, is however unfit to scale up to larger data sets and highly precise data analysis (which require many consecutive iterations of graph building process), due to its overhead in managing large data structures and its weaknesses in exploiting the full computational power of multi-core platforms.

NuChart-II is an improved C++ version of the R prototype, which has been purposely designed for performance and scalability on large datasets. It has been designed according to a structured parallel programming approach [5], [**?**]; in particular it has been designed on top of the *FastFlow* parallel programming framework, that provides high-level parallel

programming patterns for the C++ language [**?**]. The challenge was to design a software able to fully exploit the computational power exposed by multi-core architectures, making effective use of parallel processing capabilities. We have thoroughly studied the original NuChart, dividing the whole application into four main phases: 1) data retrieval from static datasets; 2) construction of the graph; 3) weighing of the edges as a result of the *normalisation* step; 4) output of results. Two main phases are suitable for being rewritten in terms of loop parallelism, since their kernels can be run concurrently on multiple processors with no data-dependencies involved: the construction of the graph and the weighing of the edges. These two phases constitute by far the most onerous parts of the application in terms of execution time: particularly when the diameter of the graph increases, these phases take up the 80% of the whole execution time.

This paper is organized as follows: section II examines the background from which this work has emerged, giving an overview of Systems Biology tools for Hi-C data analysis, together with a briefing on parallel programming techniques. Section III explains in details our work: a formal definition of our neighbourhood graph is given, and the whole application is described, focusing on the edge weighing phase that encompasses the normalisation step. Section IV describes the experiments we have conducted to test NuChart-II. Section V discusses the experiments conducted, and the data obtainable by analysing the resulting neighbourhood graphs. In this section we also describe the normalisation algorithm and discuss the performances in the edges weighing phase, obtained either using FastFlow or performed on top of other widely used parallel computing frameworks such as OpenMP and Intel TBB. Section VI concludes this paper.

II. Background

In this section we provide a comprehensive list of the available tools designed to explore and visualize Hi-C data, followed by an overview of the most commonly used and studied parallel computing frameworks.

A. Omic *Tools*

3C-based techniques used to characterize the nuclear organization of genomes and cell types have widespread among scientific communities, and a number of systems biology methods designed to analyse such data have been proposed. Particular attention is given to the detection and normalisation of systematic biases: the raw outputs of many genomic technologies are affected both by technical biases, arising from sequencing and mapping, and biological factors, resulting from intrinsic physical properties of distinct chromatin states. This makes difficult to evaluate their outcomes, as it may result in false-positives or false-negatives.

If Fluorescence *in situ* hybridization experiments are available, a good normalization solution is represented by *FisHi-Cal* [8]. This is an R package that performs an iterative *FISH*-based Hi-C calibration that exploits the information coming from both these methods. It is the first tool that integrates FISH and Hi-C data, and operates over these information to calibrate the direct measure for physical distance provided by FISH experiments and the genome-wide capture of chromatin contacts obtained by Hi-C experiments. Yaffe and Tanay [9] proposed a probabilistic model based on the observation of the genomic features. This approach can remove the majority of systematic biases, at the expense of very high computational costs, due to the observation of paired-end reads spanning all possible fragment end pairs. Hu et al [10] proposed a parametric model based on a Poisson regression. This is a simplified, and less computationally intensive normalisation procedure than the one described by Yaffe and Tanay, since it corrects the systematic biases in Hi-C contact maps at the desired resolution level, instead of modelling Hi-C data at the fragment end level. The drawback here is that the sequence information is blurred within the contact map. The first NuChart prototype [4] solved this issue by exploiting Hu et al. solution to estimate a score to each read, identifying half of the Hi-C contact instead of normalizing the contact map, thus preserving the sequence information. NuChart-II leverage this solution proposing a ex-post normalisation, that is used to estimate a probability of physical proximity between two genes, expressed as a score assigned to an edge connecting two nodes in the neighbourhood graph. Considering general software for the interpretation of Hi-C data, an interesting package is *HOMER* [11], which contains several programs and routines to facilitate the analysis of Hi-C data. Like most of the available applications, HOMER relies on the creation of contact maps for the interpretation of Hi-C data, exploiting Principal Component Analysis and hierarchical clustering with this representation. Several of the HOMER programs support multiple processors to help speed up the computation, although, while we are writing this paper, it only works at the chromosome level. *HiTC* [12] has been designed to facilitate the exploration of high-throughput, 3C-based data. It allows users to transform, normalize and visualize interaction maps. An interaction map is a two-dimensional *heat-map* representation of the matrix of Hi-C counts, whose entries correspond to the number of times two restriction fragments in a given genomic region have been ligated in 3C and sequenced as a pair. The HiTC package proposes a list of options to define the appropriate data visualization, such as contrast, color or counts trimming. *Fit-Hi-C* [13] assigns statistical confidence estimates to mid-range, intra-chromosomal contacts by jointly modelling the random polymer looping effect and previously observed technical biases in Hi-C data sets.

The visualization and exploration of Hi-C data assumes a dramatic importance when analysing Hi-C data. To the best of our knowledge, no other tool proposes a gene-centric, graph-based visualization of the neighbourhood of a gene, as NuChart does.

B. Parallel Computing Tools

Over the years, research on loop parallelism has been carried on using different approaches and techniques that vary from automatic parallelisation to iterations scheduling. In this paper we focus on the normalisation phase of NuChart-II, and compare the results obtained with FastFlow against those obtained with OpenMP and TBB, because they represent, to a major extent, the most widely used and studied frameworks for loop parallelisations.

Intel Threading Building Blocks (*TBB*) [14] is a library that enables support for scalable parallel programming using stan-

dard C++. It provides high-level abstractions to exploit task-based parallelism, independently from the underlying platform details and threading mechanisms. The TBB `parallel_for` and `parallel_foreach` methods may be used to parallelise independent invocation of the function body of a for loop, whose number of iterations is known in advance. C++11 *lambda* functions can be used as arguments to these calls, so that the loop body function can be described as part of the call, rather than being separately declared. The `parallel_for` splits the range $[0, num\_iter)$ into sub-ranges and processes each sub-range r as a separate task using a serial for loop in the code.

OpenMP [15] uses a directive based approach, where the source code is annotated with pragmas (`#pragma omp`) that instruct the compiler about the parallelism to be used in the program. In OpenMP, two constructs are used to parallelise a loop: the *parallel* and the *loop* construct. The *parallel* construct, introduced by the `parallel` directive, declares a parallel region which will be executed in parallel by a pool of threads. The *loop* construct, introduced by the `for` directive, is placed within the parallel region to distribute the loop iterations to the threads executing the parallel region. The two constructs are often fused together into the `#pragma omp parallel for` directive. OpenMP supports several strategies for distributing loop iterations among threads. The scheduling strategy may be specified via the `schedule(type[,chunk size])` clause, which is appended to the `for` directive. The `type` of scheduling policy can be one among *static*, *dynamic*, *guided*, *auto*, *runtime*, each one providing a different thread scheduling policy.

FastFlow[1] is a parallel programming environment originally designed to support efficient streaming on cache-coherent multi-core platforms and distributed systems [**?**], [**?**]. It is realised as a C++, pattern-based, parallel programming framework, aimed at simplifying the development of applications for multi-core and GPGPUs platforms. It provides developers with a set of high-level, parallel programming patterns (aka *algorithmic skeletons*), obtained by the composition of two basic algorithmic skeletons: a *farm* skeleton, and a *pipeline* skeleton. Leveraging the *farm* skeleton, FastFlow exposes a `ParallelFor` pattern [16], where farm workers are sequential wrappers that execute chunks of a loop iterations having the form `for(idx=start;idx<stop;idx+=step)`. Just like TBB, FastFlow's `ParallelFor` pattern uses C++11 *lambda* functions as a concise and elegant way to create a function object: lambdas can "capture" the state of non-local variables by value or by reference and allow functions to be syntactically defined when needed.

III. Genes Neighbourhood Graph

We recall that a graph is a formal mathematical representation of a collection of vertices (V) connected by edges (E), which model a relationship among vertices. In this context, vertices represent genes. We say that two genes are *connected* if there exists paired-end Hi-C data belonging to both of them (Figure 1). We define this paired-end Hi-C data as a *connection*, meaning a spatial relationship between genes. A more formal definition of our graph can be expressed as:

$$G = (V, E)$$
$$\textbf{where}$$
$$V = \{\ g \mid g \in \textit{Genes}\ \}$$
$$\textbf{and}$$
$$E = \{\ (g_1, g_2) \mid g_1 \rightleftharpoons g_2\ \wedge\ g_1, g_2 \in V\ \}.$$

The resulting graph G is an *undirected, weighted* graph with a symmetric binary relation between the adjacent vertices – i.e., if g_1 is connected to g_2, then g_2 is connected to g_1 – while the weights on the edges provide a likelihood of physical proximity for the adjacent vertices, as a result of the normalisation phase. The neighbourhood graph can be defined as the induced *subgraph* obtainable starting from a given root vertex v, and including all vertices adjacent to v and all edges connecting such vertices, including the root vertex. Despite being an undirected graph, it is not a *simple* graph: self-loops are allowed to exist. This happens because the paired end of a gene's fragment might be extremely close to the first end, so that both fragments belong to the same gene, thus forming a self-loop. Upon these formal basis, our neighbourhood graph represents a topological map of the specific nucleus region to which a gene belongs.

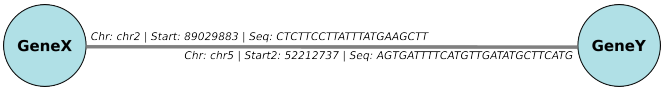

Fig. 1 – *Genes are the vertices of our graph, while edges represent paired-ends fragments that belong to the adjacent genes*

A. NuChart-II

NuChart-II is an improved C++ version of the first R prototype. In the development of this new version of the software, particular attention has been paid at optimising the data structures employed for the construction of the graph, in order to facilitate the parallel implementation of the algorithm. This novel implementation refines the normalisation routine, which relies on a modified version of the Hu et al. approach [10]: while the original prototype used the Poisson regression model to provide a score to each read, NuChart-II exploits the same regression analysis to assign a confidence score to each edge of the neighbourhood graph, so that the user can evaluate the reliability of each contact. The engineering of the new software has been conducted on top of *FastFlow*, using the `ParallelFor` pattern discussed above (see section II-B): FastFlow aims at simplifying the programmers life in developing complex parallel applications, while providing high runtime efficiency. Both the graph construction and the normalisation routine have been tested against OpenMP's `parallel for` and Intel TBB's `parallel_for`.

Graph Construction: starting from one or more root nodes, a graph of adjacent genes is constructed. Neighbours identification begins by searching those chromosome fragments belonging to the starting gene(*s*). These fragments are then compared with other chromosome fragments located in a different genomic region, annotated in the coupled reads. When a match is found and the new fragment overlaps a gene's

[1]FastFlow is an open source project: http://mc-fastflow.sourceforge.net/

region, an edge between the starting gene and the novel detected one is created.

NuChart-II can also handle the case of inter-genic contacts: if the identified chromosome fragment is inter-genic, the corresponding genomic position is represented on the graph as a singularity point. These singularity points may be expanded to the closest nearby genes (*after* and *before* genes). Note that singularity points do not belong to the genes domain, as they merely represent a position along the chromosome fragment expressed in terms of coordinates. Their graphical representation differs from other genes (Figure 2).

Neighbours can be searched up to the desired "distance" from the root, that determines the levels of the resulting graph: a search at level 1 yields all the genes directly adjacent to the root (which is at level 0); a search at level i returns all directly adjacent genes for each gene discovered up to level $i-1$, starting from the root.

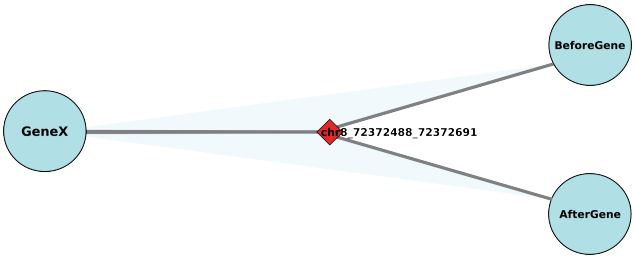

Fig. 2 – *Singularity points not belonging to the vertices domain have a different graphical representation*

Taking inspiration from the work of Hong et al. [17], the algorithm for graph construction proceeds as *Breadth First Search*. At each iteration level, it collects all the candidate connections in the current BFS level (by comparing reads and genes coordinates), then searches for the genes' neighbours, in parallel, over all the connections. At the end of each level iteration, the parallel execution is synchronized: at this point thread-local next-level containers are processed and a *partial* graph is constructed with the nodes discovered at the current BFS level. The definitive graph is built in batch at the end of the BFS execution. The iterations proceed until all the nodes of the graph have been visited, or preferably up to the desired level. Algorithm 1 reports a pseudo-code for the graph construction.

This high-level approach required some adjustment to the BFS procedure and the introduction of new thread-local containers, needed to handle concurrent write accesses to shared data structures. Specifically, C[NTH], V[NTH] and E[NTH] are used to store per-thread data, where NTH is the number of threads in use and *thid* identifies thread's own container, such that $0 \leq thid < \text{NTH}$. Q represents our working queue that contains the genes discovered throughout the computation. *L_MAX* determines the maximum distance from the root that has to be reached (-1 means explore all graph). Γ is used at every level synchronisation to store partial graphs, that will be merged into a definitive graph at the very end of the graph construction process.

Concerning data structures and memory management, the objects involved in the computations contain a considerable amount of information that burden the working set with lots of unused data. This undesirable effect has been prevented by duplicating only the needed resources, resulting in a reduced working set that helps minimising the cache thrashing and permits to obtain substantial performance improvements.

Algorithm 1 *Graph Construction*

```
BuildNeighbourhoodGraph (root, L_MAX, NTH) {
   Q = Γ = Graph := ∅
   C[NTH] = V[NTH] = E[NTH] := ∅
   lv := 0

   push root in Q
   while (Q not ∅ and lv < L_MAX) {
      pop q from Q
      // find Hi-C Reads for q
      ParallelFor (r in Reads, NTH) {
         if (r.Start in q[Start, Stop] and r.Chr == q.Chr)
            add r to C[thid]
      }
      // find neighbour genes for q
      ParallelFor (c in C[thid], NTH) {
         intra := 0
         for_each (g in Genes) {
            if (g overlaps c.PairedEnd)
               add g to V[thid]
               add (q, g) to E[thid]
               intra := intra + 1
         }
         HandleIntergenicCase(Genes,intra)
      }
      // level synchronisation
      Γ := BuildPartialGraph(V[thid],E[thid])

      for_each (thid in [0, NTH-1]) {
         for_each (v in V[thid]) { // next level vertices
            if (not v.Visited)
               push v in Q
         }
      }
      lv := lv + 1
      C[thid] = V[thid] = E[thid] := ∅
   }
   Graph := BuildGraph(Γ)
}
```

Edge Weighing: this task encompasses the normalisation process, which is needed in order to remove systematic biases arising from sequencing and mapping. The weight assumes the role of a "confidence score" that characterize the reliability of each contact represented on the neighbourhood graph. We recall that an edge identifies the existence of Hi-C fragments belonging to both connected genes; for each edge, a contact map (M) is constructed directly modelling the read count data at a resolution level of 1 MB. Hi-C data matrix is symmetric, thus we consider only its upper triangular part, where each point of $M_{i,j}$ denotes the intensity of the interaction between positions i and j. Using the local genomic features that describe the chromosome (fragment length, GC-content and mappability), we can set up a *generalized linear model* (GLM) with Poisson regression, with which we estimate the maximum likelihood of the model parameters. The model is given by the formula:

$$e(Y|X) = g\{X^T\beta\}.$$

Here β denotes the parameter vector to be estimated and g denotes a known link function. The contact map incorporates the information about the independent variables of our model (i.e. the expected value $\mu = e(Y|X)$); chromosome length and GC-content act as regressors (i.e. the coefficients of the

linear combination $g(X^T\beta)$. This model is used to count the occurrences in a fixed amount of space: for this reason we choose the Poisson distribution, and the natural logarithm as the link function for our model. With the best-fit coefficients returned by the linear regression the score is computed, so that the edge contains an estimate of the physical proximity, plus the genomic information for both genes, which are preserved and not blurred within the contact map.

The generalized linear model with Poisson regression has been implemented adapting the *Iteratively Weighted Least Squares* algorithm (IWLS) proposed by Nelder and Wedderburn [18]. The listing below reports a pseudo-code of the function:

Algorithm 2 *Edge Weighing*

```
ComputeEdgeScore(edge, τ) {
   LenM = GccM = MapM := ∅  // cover matrices
   X = Y = B := ∅
   Conv := true

   // populate cover matrices using genomic features
   ...

   X := ToMatrix(LenM, GccM)
   Y := BuildContactMap(edge.Chr1, edge.Chr2)

   while (Conv) {
      ApplyLinkFunction(Y)
      B := ApplyGLM(Y, X, MapM)
      Conv := CheckConvergence(B, τ)
   }

   edge.Score := f(B)
}
```

Fragment length and GC-content are combined to form the linear predictor matrix for the GLM model. The mappability feature, stored within the *MapM* matrix, is used as an offset for the linear regression. The regression is run until a convergence criterion is met: in our case we check that the absolute value of the χ^2 (*chi-squared*) difference at each iteration is less than a certain threshold τ:

$$|\chi^2 - \chi^2_{old})| < \tau.$$

Algorithm 2 shows a pseudo-code of the weighing algorithm: the function `ApplyGLM` writes the best-fit parameters in vector B, which is the result of the regression: these coefficients are used to calculate the score (i.e. the estimation of physical proximity) for the edge connecting the two genes. Also, we compute *dispersion* and *standard error*, so as to provide a useful summary of model fit.

IV. Experiments

NuChart-II has been designed to overcome the weaknesses of the R prototype, which had significant bottlenecks in memory management and limitations in the exploitation of the available computational resources, causing restrictions in the usability of the tool. This novel implementation addresses these weaknesses, making possible a genome-wide exploration of Hi-C contacts – thanks to the optimal memory management and data structure design – with outstanding improvements in terms of execution time, obtained exploiting loop parallelism techniques on multi-core architectures. We have conducted a number of experiments to verify correctness and goodness of NuChart-II: starting from the work in [4] we have replicated some of the tests conducted there, in order to have a basis for comparing the accuracy of the results. We have increased the number of iterations to further explore genes' neighbourhood, while also testing the novel tool on bigger datasets.

As target architecture we considered a NUMA Intel workstation equipped with 4 eight-core E7-4820 Nehalem running at 2.0GHz, featuring 18MB L3 cache per NUMA node, 256KB L2 cache and 64KB L1 cache (L1d + L1i), with 64 GB of main memory. The Nehalem processor uses HyperThreading with 2 contexts per core. We use up to 32 threads in order to exploit all physical cores without making use of the second context. Thanks to the internal structure of the FastFlow *ParallelFor*, it is possible to use all physical cores while thread pinning is automatically managed by the FastFlow library. We used the GNU gcc 4.8.0 compiler with the optimisation flag -03. As a performance metric, together with the overall execution time, we used also the speedup, calculated as

$$S = T(seq)/T(n)$$

where $T(seq)$ is the sequential execution time measured during a plain sequential execution, and $T(n)$ is the parallel execution time using n worker threads, keeping the problem size fixed.

DNA Exploration

We performed several executions involving the creation of neighbourhood graphs for relevant genes or gene clusters, at different levels of iterations, in order to demonstrate the accuracy and usability of NuChart-II. The tests have been conducted either considering intra-genic contacts only, or expanding also inter-genic contacts. LiebermanAiden et al. Hi-C experiment SRA:SRR027963 [3] and Dixon et al. Hi-C experiment SRA:SRR400262 [19] are the datasets used as a test-bed for the genome exploration. We used chromosome fragments obtained with *HindIII* as digesting enzyme .

Figure 3 shows a neighbourhood graph for the gene TP53 at level 1 (left) and level 2 (right), according to the LiebermanAiden Hi-C experiment. In figure 4, also the inter-genic contacts have been expanded for the same gene TP53. The root gene is yellow coloured. In figure 3 (and 4) two direct neighbours of the root gene TP53 exhibit a high degree of connected components: the role of *hub* that the two genes (KIA0753 and PHF2) acquire, suggest their importance in maintaining the interactions in that particular genomic region.

A drawback of this visualization is that the readability is dramatically compromised when the number of nodes and edges increases, likely resulting in a tangle of edges hardly understandable. NuChart-II supports plotting with *iGraph* and *GraphViz*: these tools perform nicely with small-to-medium sized graphs, but cannot provide useful representation of huge graphs with more than ten thousand edges (as it happens when the diameter of the graph increases). Textual and tabular outputs become useful for the analysis of the genomic regions explored: the probability of a connection can be estimated by evaluating an edge's weight, while the overall graph structure is shown in terms of the distance of each discovered gene from the root(s).

Also, we are investigating possible approaches to detect communities and clusters, which may help in giving alternative representations of the graph and may open new perspectives on the Hi-C data analysis and interpretation.

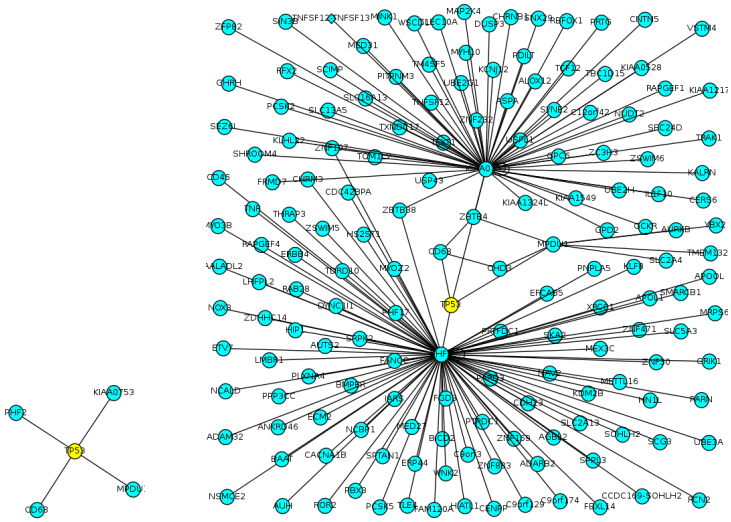

Fig. 3 – *Neighbourhood graph for gene **TP53** according to LiebermanAiden experiment, with neighbours at Level 1 (*left*) and Level 2 (*right*)*

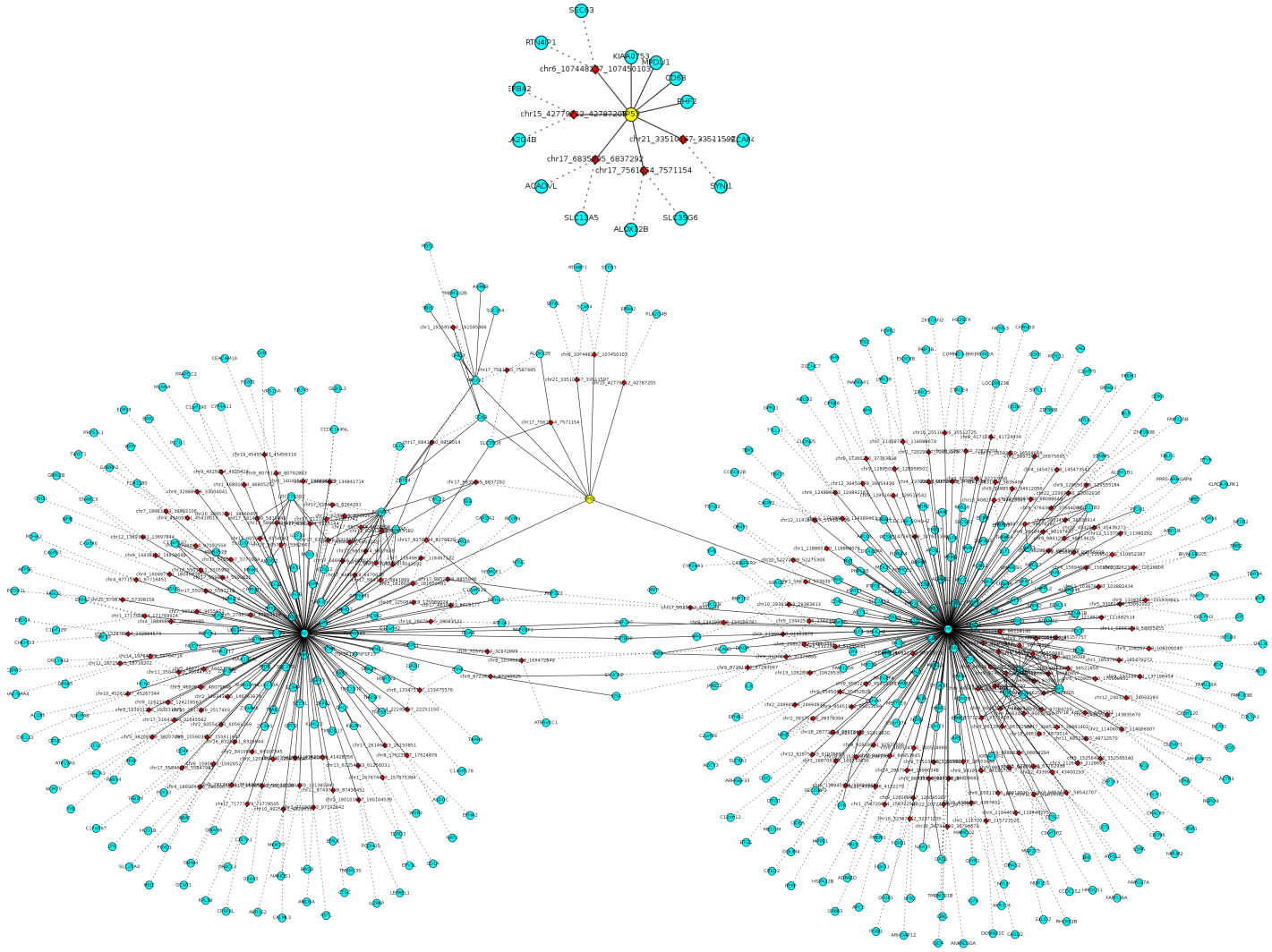

Fig. 4 – *Neighbourhood graph for gene **TP53** according to LiebermanAiden experiment, with **inter-genic** expanded. Neighbours at Level 1 (*up*) and Level 2 (*down*)*

V. Discussion

Network Analysis and Statistics

The graph-based approach opens new perspectives on the study of the 3D chromosome conformation and the genes interaction: the *social network* point of view allows to study the relationships among genes in terms of network theory. NuChart-II performs statistical measures over the graphs and produces additional outputs useful to examine the results: starting with a centrality index analysis, it is possible to the identify the most important and influential genes. In this particular context, the node degree distribution leads us to consider the genes network as a scale-free network, which suggests further investigations towards detecting the presence of community structures. This network perspective permits to shed some new light over the genes interaction.

Performance

Both the graph construction and the edges weighing phase are bounded to the memory size required to hold the data. We have accurately tuned the crucial steps in order to maximize the use of memory hierarchy and fully exploit cache locality, while minimising cache trashing.

Details of the graph construction phase have been presented elsewhere. The edges weighing phase is an embarrassingly parallel application: any arbitrary subset of the edges can be processed independently from each other by mean of a parallel loop pattern. With Fastflow's `ParallelFor` this data-parallelism can be properly exploited to boost up performances and drastically reduce execution time. This can be accomplished by simply defining our weighing kernel as the lambda function of the `ParallelFor`.

During execution, each worker thread gets a bunch of edges to work on, according to the grain size: we have found that the best performances are reached when the grain size is purposely kept small. Each thread uses three thread-local read-only static data structures that hold information about local genomic features. These data are used to build all matrices needed to construct the regression workspace. The task involves tight loops doing Floating Point arithmetic calculations on data that fit the L3 cache and can fully benefit from compiler optimizations and vectorization. On the other hand, a number of dynamic memory allocations are necessary during the execution of the normalisation step. The use of a memory allocator not designed for parallel programming causes a serialization of the operations that leads to a reduction of the total execution time.

Despite the large memory footprint, the implementation with FastFlow shows a quasi-ideal speedup: the memory intensive computations performed hide the latency to memory accesses, and when compared against OpenMP and Intel TBB, the recorded performances are substantially similar (figure 5). Intel TBB begins to suffer for the dynamic memory allocations when the number of threads is greater than 24, causing its performance to flatten.

Tests have been conducted using as much similar configurations as possible, trying either with static scheduling or with dynamic scheduling and variable chunk size. With Intel TBB's `parallel_for`, we used the *affinity partitioner*, as it attempts to perform some automatic cache optimizations, although it did not bring substantial improvements with respect to the default *auto partitioner*. With FastFlow's `ParallelFor` we found that the best performances were reached when using the scheduler as thread. OpenMP `parallel for` produced the best performance with dynamic scheduling.

We have conducted our tests with a maximum of 31 cores in our machine with 32 cores (excluding two-ways hyper-threading): when using FastFlow's `ParallelFor` the scheduler can be adapted to be run as thread or as an object. when the former solution is chosen, the number of running threads is $\#worker\_threads + 1$. When this number equals the number of cores, the extra thread used (which performs busy-waiting during synchronisation) introduces non negligible overhead, especially in fine grain computations. We have anyway noticed that this configuration yields more desirable results in terms of overall performance.

VI. Conclusions and Future Works

The novel implementation of NuChart-II allows the software to scale genome-wide, which is crucial to exploit its full capability for a correct analysis, interpretation and visualisation of the chromosome conformation. This graph-based approach opens new perspectives for the analysis and processing of Hi-C data, focusing more on the interactions of a gene with its neighbourhood. Furthermore, the normalisation phase has been revisited and provides a valuable estimate of physical proximity for two genes, while keeping available all genomic data related to the spatial region where the genes lie. We have shown that such genome-wide exploration and analysis is possible with the aid of novel high-level parallel programming patterns, that allow to address many of the issues that burdened the original R prototype, obtaining performances that would have been inconceivable with the original R prototype. As for the future works, we are investigating practical and convenient solutions to address the visualisation problem: starting from community structures detection, the graph visualization can be ameliorated by providing a representation at different scales: when the scale is small only communities and their connections are shown, while as the scale grows genes must become visible and readable. We aim at building dynamic, interactive graphs where all the physical, chemical and statistical information are easily accessible by direct interaction with the graph. We are investigating the benefits obtainable when using a scalable, lock-free memory allocator. We are also considering a possible GPGPU implementation of the weighing phase, where the huge computational power exposed by modern GPU devices can be used to reduce the execution time of heavy mathematical calculations.

Acknowledgement

This work has been partially supported by the EC-FP7 STREP project Paraphrase (no. 288570), the EC-FP7 STREP project REPARA (no. 609666) and the Fondazione San Paolo IMPACT project (ID. ORTO11TPXK).

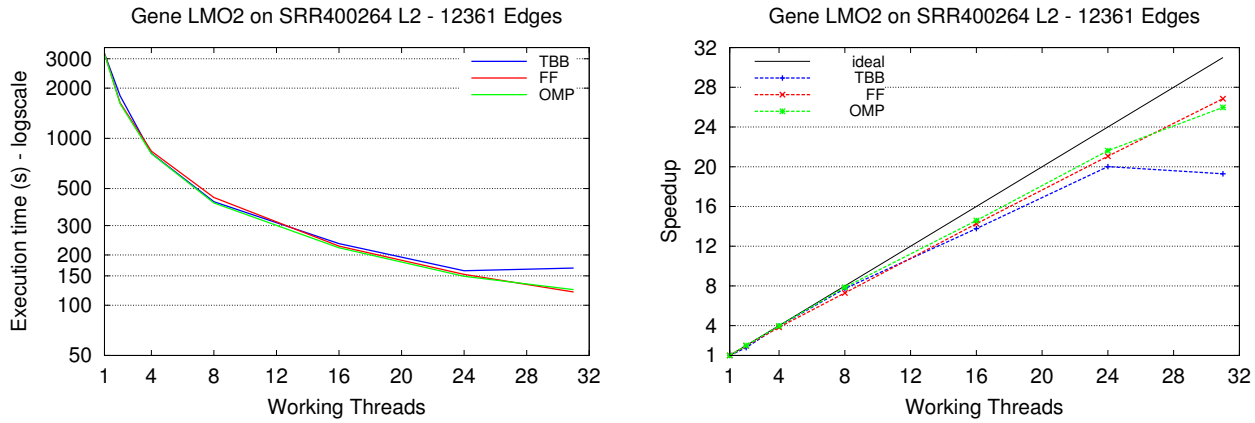

Fig. 5 – *Execution time (left) and speedup (right) of the normalisation phase for 12361 edges, resulting from a Level 2 neighbourhood graph for the gene LMO2, according to the Dixon et al. SRA:SRR400264 experiment*

REFERENCES

[1] E. de Wit and W. de Laat, "A decade of 3C technologies: insights into nuclear organization," *Genes & Development*, vol. 26, no. 1, pp. 11–24, Jan. 2012.

[2] J. Dekker, K. Rippe, M. Dekker, and N. Kleckner, "Capturing Chromosome Conformation," *Science*, vol. 295, no. 5558, pp. 1306–1311, 2002.

[3] E. Lieberman-Aiden, N. L. van Berkum, L. Williams, M. Imakaev, and T. e. a. Ragoczy, "Comprehensive mapping of long-range interactions reveals folding principles of the human genome," *Science*, vol. 326, no. 5950, pp. 289–293, 2009.

[4] I. Merelli, P. Li, and L. Milanesi, "NuChart: An R Package to Study Gene Spatial Neighbourhoods with Multi-Omics Annotations," *PLoS ONE*, vol. 8, no. 9, Sep. 2013.

[5] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, 2009.

[6] M. Aldinucci, S. Campa, M. Danelutto, P. Kilpatrick, and M. Torquati, "Design patterns percolating to parallel programming framework implementation," *International Journal of Parallel Programming*, Sep. 2013.

[7] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, "Fastflow: high-level and efficient streaming on multi-core," in *Programming Multi-core and Many-core Computing Systems*, ser. Parallel and Distributed Computing, S. Pllana and F. Xhafa, Eds. Wiley, 2014, ch. 13.

[8] Y. Shavit, F. Hamey, and P. Lió, "FisHiCal: an R package for iterative FISH-based calibration of Hi-C data," *Bioinformatics*, vol. 30, no. 18, Sep. 2014.

[9] A. T. Eitan Yaffe, "Probabilistic modeling of Hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture," pp. 1059–1065, 2011.

[10] M. Hu, K. Deng, S. Selvaraj, Z. Qin, B. Ren, and J. S. Liu, "HiCNorm: removing biases in Hi-C data via Poisson regression." *Bioinformatics (Oxford, England)*, vol. 28, no. 23, pp. 3131–3133, Dec. 2012.

[11] V. C. Seitan, A. J. Faure, Y. Zhan, R. P. P. McCord, B. R. Lajoie, E. Ing-Simmons, B. Lenhard, L. Giorgetti, E. Heard, A. G. Fisher, P. Flicek, J. Dekker, and M. Merkenschlager, "Cohesin-based chromatin interactions enable regulated gene expression within preexisting architectural compartments." *Genome research*, vol. 23, no. 12, pp. 2066–2077, Dec. 2013.

[12] N. Servant, B. R. Lajoie, E. P. Nora, L. Giorgetti, C.-J. Chen, E. Heard, J. Dekker, and E. Barillot, "HiTC: exploration of high-throughput 'C' experiments," *Bioinformatics*, vol. 28, no. 21, pp. 2843–2844, Nov. 2012.

[13] F. Ay, T. Bailey, and W. Noble, "Statistical confidence estimation for Hi-C data reveals regulatory chromatin contacts," 2014.

[14] "Intel Threading Building Blocks, project site," 2013, http://threadingbuildingblocks.org.

[15] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[16] M. Danelutto and M. Torquati, "Loop parallelism: a new skeleton perspective on data parallel patterns," in *Proc. of Intl. Euromicro PDP 2014: Parallel Distributed and network-based Processing*, M. Aldinucci, D. D'Agostino, and P. Kilpatrick, Eds. Torino, Italy: IEEE, 2014. [Online]. Available: http://calvados.di.unipi.it/storage/paper_files/2014_ff_looppar_pdp.pdf

[17] S. Hong, T. Oguntebi, and K. Olukotun, "Efficient parallel graph exploration on multi-core cpu and gpu," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 78–88. [Online]. Available: http://dx.doi.org/10.1109/PACT.2011.14

[18] J. A. Nelder and R. W. M. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society, Series A, General*, vol. 135, pp. 370–384, 1972.

[19] J. Dixon, S. Selvaraj, F. Yue, A. Kim, Y. Li, Y. Shen, M. Hu, J. Liu, and B. Ren, "Topological domains in mammalian genomes identified by analysis of chromatin interactions." *Nature*, vol. 485, no. 5, pp. 376–80, 2012.